[卷积 3](#_Toc999577826)

[1、基本概念 3](#_Toc967096403)

[（1）卷积翻转 3](#_Toc1149880995)

[（2）CNN模型 6](#_Toc1540685500)

[（3）理解卷积和子采样 6](#_Toc1945905300)

[（4） 理解用卷积代替全连接 6](#_Toc2119407459)

[（5）通过基本的神经元模型理解可训练参数与连接数 7](#_Toc919086512)

[2、卷积神经网络CNN 8](#_Toc1901720675)

[（1）局部感知域 8](#_Toc174643247)

[（2）共享权值 10](#_Toc937818200)

[（3）池化 13](#_Toc82623750)

[3、详解CNN的层次结构 16](#_Toc1949609836)

[（1）输入层 16](#_Toc1392302148)

[（2）卷积层 18](#_Toc46760326)

[（3）激励层 22](#_Toc1952079547)

[（4）池化层 23](#_Toc1671886597)

[（5）归一化层 24](#_Toc2022750713)

[（6）切分层： 27](#_Toc1758248908)

[（7）融合层： 27](#_Toc144254571)

[（8）全连接层和输出层 28](#_Toc340458160)

[4、CNN训练过程 29](#_Toc1667283918)

[4.1神经网络BP算法 29](#_Toc1297814483)

[4.2 BP算法在CNN中的应用 31](#_Toc1633337247)

[5、典型卷积网络 37](#_Toc1919642164)

[（1）LeNet-5 38](#_Toc271660947)

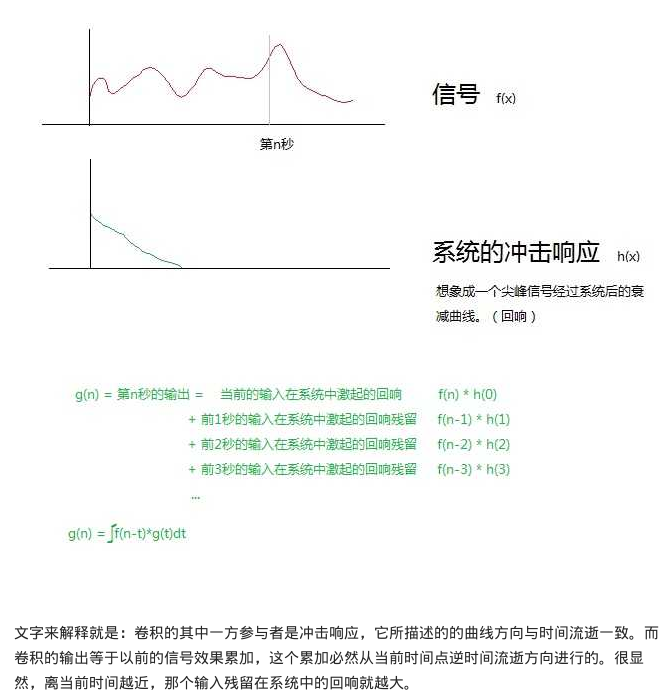
[（2）VGG 44](#_Toc1026862333)

# 卷积

## 1、基本概念

### （1）卷积翻转

参考网址：<https://www.zhihu.com/question/20500497>



时间差f（x）和信号强度h（y）。以当前时间n为初始点，此时的信号强度为h（0），然后去观察n-1时刻的信号强度，相当于站在现在看过去的未来，过去是n-1，它此时的强度是最强的，也可以理解成h（0），那么未来的强度会比它弱（因为是衰减函数），也就是h（1）。

更为抽象一点的解释：两个函数，翻转其中一个，再滑动求积分，叫卷积（convultion）；不翻转就滑动求积分，叫做互相关（cross-correlation）。如果其中之一是偶函数，那么卷积和互相关效果相同。从定义上看，翻转这个操作就是一步操作而已，具体的物理意义只能在应用中找到。

以信号处理为例，卷积意味着把输入信号在时间轴上翻转，然后跟信号处理系统的描述方程（冲激响应）叠加积分。为什么要翻转？因为这样才符合现实：输入信号的 0 秒先跟冲激响应的 0 秒叠加，然后输入信号的 1 秒和冲激响应的 1 秒叠加，以此类推。当你把这两个函数分别画出来上下并列的时候，它们就好象合并的拉链，0 点处在同一侧，而卷积实际上是要把它们画在同一个轴上滑动，同时却必须保证输入信号的 0 点先遇到冲激响应函数的 0 点——怎么办呢？就好像拉链被拉开了：翻转一下。

MXnet是亚马逊开发的深度学习框架

上采样和下采样

缩小图像（或称为下采样（subsampled）或降采样（downsampled））的主要目的有两个：1、使得图像符合显示区域的大小；2、生成对应图像的缩略图。

放大图像（或称为上采样（upsampling）或图像插值（interpolating））的主要目的是放大原图像,从而可以显示在更高分辨率的显示设备上。

对图像的缩放操作并不能带来更多关于该图像的信息, 因此图像的质量将不可避免地受到影响。然而，确实有一些缩放方法能够增加图像的信息，从而使得缩放后的图像质量超过原图质量的。

下采样原理：对于一副图像I尺寸为M\*N，对起进行s倍下采样，即得到（M/s）\*（N/s）尺寸的分辨率图像，当然，s应该是M和N的公约数才可以，如果考虑是矩阵形式的图像，就是把原始图像s\*s窗口内的图像编程一个像素，这个像素点的值就是窗口内所有像素的均值。

上采样原理：图像放大几乎都是采用内插值方法，即在原有图像像素的基础上在像素点之间采用合适的插值算法插入新的元素。

插值算法还包括了传统插值，基于边缘图像的插值，还有基于区域的图像插值。

CNN学习

对图像来说，总思路是将图片不断进行压缩，直到输出单一的值。

C语言实现CNN

<http://blog.csdn.net/tostq/article/details/51786265>

基于Tensorflow框架的CNN--python实现

<http://blog.csdn.net/xukaiwen_2016/article/details/70880694> <https://zhuanlan.zhihu.com/p/26645110?refer=carefree0910-pyml>

基于theano框架的CNN--Python实现

<http://blog.csdn.net/u014365862/article/details/54865609>

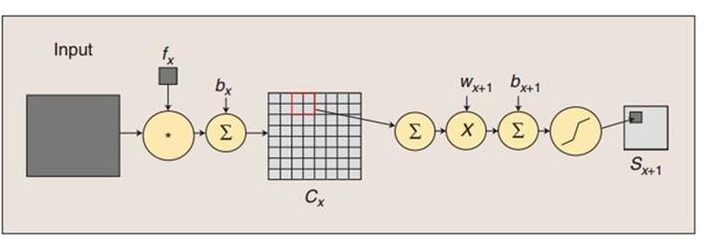
C++实现的深度学习框架--->tiny-cnn

<http://blog.csdn.net/column/details/tiny-cnn.html>

### （2）CNN模型



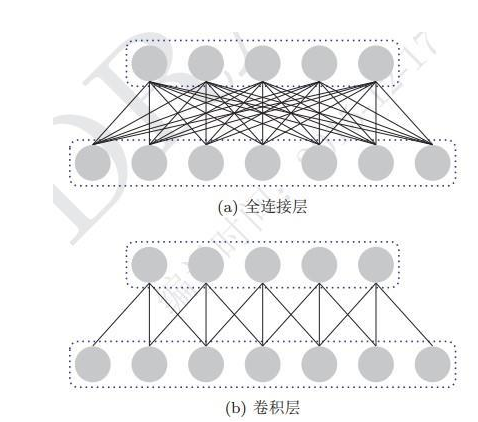
### （3）理解卷积和子采样



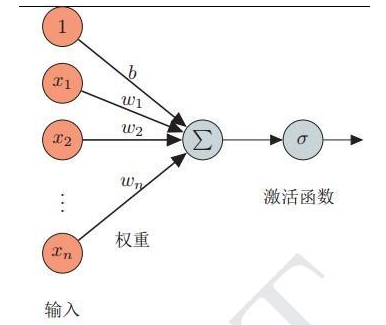
卷积过程：用一个可训练的滤波器fx去卷积一个输入的图像（第一阶段是输入的图像，后面的阶段就是卷积特征map），然后加一个偏置bx，得到卷积层Cx;

子采样过程：邻域四个像素求和变为一个像素，然后通过标量W加权，再增加偏置b，然后通过一个sigmoid激活函数，产生一个缩小四倍的特征映射图Sx+1

### （4） 理解用卷积代替全连接



### （5）通过基本的神经元模型理解可训练参数与连接数

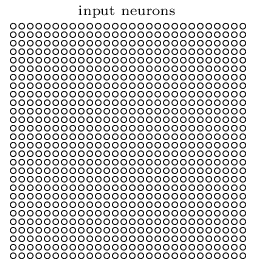


## 2、卷积神经网络CNN

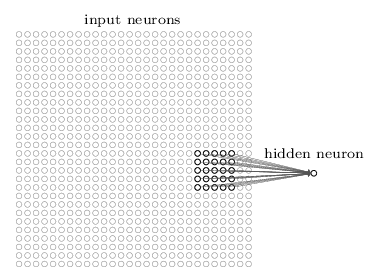
卷积神经网络有3个基本的idea：局部感知域(Local Recpetive Field)，权值共享和池化(Pooling)。下面我们来一个一个的介绍它们。

### （1）局部感知域

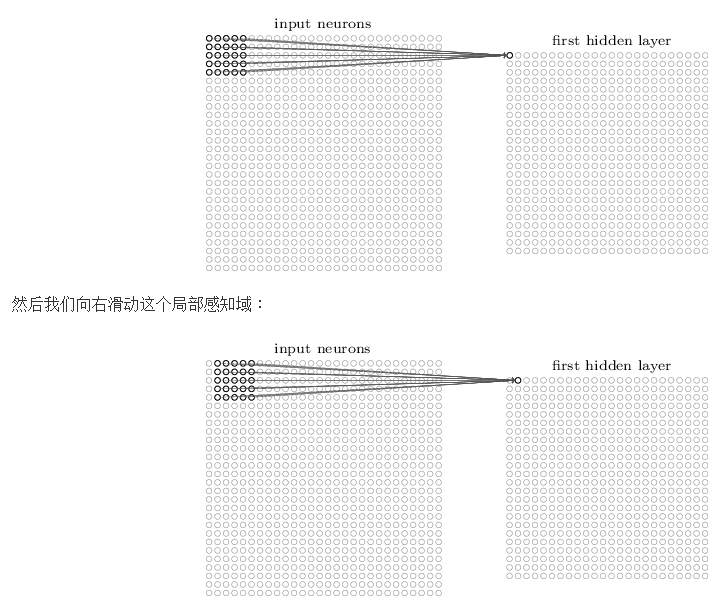
在前面图示的全连接的层里，输入是被描述成一列神经元。而在卷积网络里，我们把输入看成28×28方格的二维神经元，它的每一个神经元对应于图片在这个像素点的强度（灰度值），如下图所示：



和往常一样，我们把输入像素连接到隐藏层的神经元。但是我们这里不再把输入的每一个像素都连接到隐藏层的每一个神经元。与之不同，我们把很小的相临近的区域内的输入连接在一起。更加具体的来讲，隐藏层的每一个神经元都会与输入层一个很小的区域（比如一个5×5的区域，也就是25个像素点）相连接。隐藏对于隐藏层的某一个神经元，连接如下图所示：



输入图像的这个区域叫做那个隐藏层神经元的局部感知域。这是输入像素的一个小窗口。每个连接都有一个可以学习的权重，此外还有一个bias。你可以把那个神经元想象成用来分析这个局部感知域的。我们然后在整个输入图像上滑动这个局部感知域。对于每一个局部感知域，都有一个隐藏层的神经元与之对应。为了具体一点的展示，我们首先从最左上角的局部感知域开始：



以此类推，我们可以构建出第一个隐藏层。注意，如果我们的输入是28×28，并且使用5×5的局部关注域，那么隐藏层是24×24。因为我们只能向右和向下移动23个像素，再往下移动就会移出图像的边界了。【说明，后面我们会介绍padding和striding，从而让图像在经过这样一次卷积处理后尺寸可以不变小】

这里我们展示了一次向右/下移动一个像素。事实上，我们也可以使用一次移动不止一个像素【这个移动的值叫stride】。比如，我们可以一次向右/下移动两个像素。在这篇文章里，我们只使用stride为1来实验，但是请读者知道其他人可能会用不同的stride值。

### （2）共享权值

之前提到过每一个隐藏层的神经元有一个5×5的权值。这24×24个隐藏层对应的权值是相同的。也就是说，对于隐藏层的第j,k个神经元，输出如下：

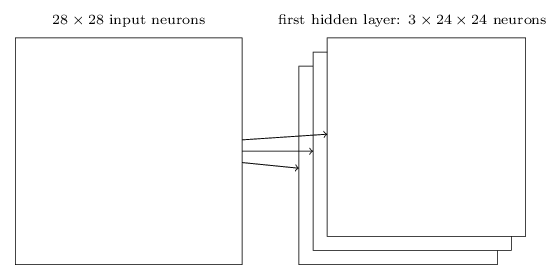
这里，σ是激活函数，可以是我们之前提到的sigmoid函数。b是共享的bias，Wl,m 是5×5的共享权值。ax,y 是输入在x,y的激活。

【从这个公式可以看出，权值是5×5的矩阵，不同的局部感知域使用这一个参数矩阵和bias】

这意味着这一个隐藏层的所有神经元都是检测同一个特征，只不过它们位于图片的不同位置而已。比如这组weights和bias是某个局部感知域学到的用来识别一个垂直的边。那么预测的时候不管这条边在哪个位置，它都会被某个对于的局部感知域检测到。更抽象一点，**卷积网络能很好的适应图片的位置变化**：把图片中的猫稍微移动一下位置，它仍然知道这是一只猫。

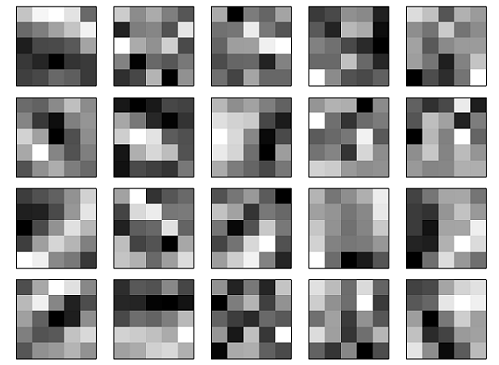
因为这个原因，我们有时把输入层到隐藏层的映射叫做特征映射(feature map)。我们把定义特征映射的权重叫做共享的权重(shared weights)，bias叫做共享的bias(shared bais)。这组weights和bias定义了一个kernel或者filter。

上面描述的网络结构只能检测一种局部的特征。为了识别图片，我们需要更多的特征映射。隐藏一个完整的卷积神经网络会有很多不同的特征映射：



在上面的例子里，我们有3个特征映射。每个映射由一个5×5的weights和一个biase确定。因此这个网络能检测3种特征，不管这3个特征出现在图像的哪个局部感知域里。

为了简化，上面之展示了3个特征映射。在实际使用的卷积神经网络中我们会使用非常多的特征映射。早期的一个卷积神经网络——LeNet-5，使用了6个特征映射，每一个都是5×5的局部感知域，来识别MNIST数字。因此上面的例子和LeNet-5很接近。后面我们开发的卷积层将使用20和40个特征映射。下面我们先看看模型学习到的一些特征：



这20个图片对应了20个不同的特征映射。每个映射是一个5×5的图像，对应于局部感知域的5×5个权重。颜色越白(浅)说明权值越小(一般都是负的)，因此对应像素对于识别这个特征越不重要。颜色越深(黑)说明权值越大，对应的像素越重要。

那么我们可以从这些特征映射里得出什么结论呢？很显然这里包含了非随机的空间结构。这说明我们的网络学到了一些空间结构。但是，也很难说它具体学到了哪些特征。我们学到的不是一个 Gabor滤波器 的。事实上有很多研究工作尝试理解机器到底学到了什么样的特征。如果你感兴趣，可以参考Matthew Zeiler 和 Rob Fergus在2013年的论文 Visualizing and Understanding Convolutional Networks。

共享权重和bias的一大好处是它极大的减少了网络的参数数量。对于每一个特征映射，我们只需要 25=5×5 个权重，再加一个bias。因此一个特征映射只有26个参数。如果我们有20个特征映射，那么只有20×26=520个参数。如果我们使用全连接的神经网络结构，假设隐藏层有30个神经元(这并不算很多)，那么就有784\*30个权重参数，再加上30个bias，总共有23,550个参数。换句话说，全连接的网络比卷积网络的参数多了40倍。

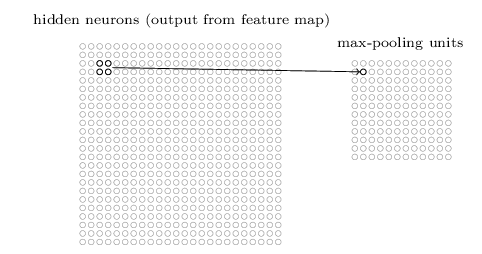
当然，我们不能直接比较两种网络的参数，因为这两种模型有本质的区别。但是，凭直觉，由于卷积网络有平移不变的特性，为了达到相同的效果，它也可能使用更少的参数。由于参数变少，卷积网络的训练速度也更快，从而相同的计算资源我们可以训练更深的网络。

“卷积”神经网络是因为公式(1)里的运算叫做“卷积运算”。更加具体一点，我们可以把公式(1)里的求和写成卷积：$a^1 = \sigma(b + w \* a^0)$。\*在这里不是乘法，而是卷积运算。这里不会讨论卷积的细节，所以读者如果不懂也不要担心，这里只不过是为了解释卷积神经网络这个名字的由来。【建议感兴趣的读者参考colah的博客文章 《Understanding Convolutions》】

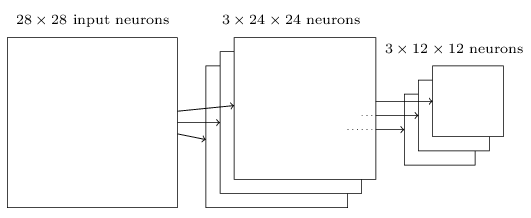
### （3）池化

除了上面的卷积层，卷积神经网络也包括池化层(pooling layers)。池化层一般都直接放在卷积层后面池化层的目的是简化从卷积层输出的信息。

更具体一点，一个池化层把卷积层的输出作为其输入并且输出一个更紧凑(condensed)的特征映射。比如，池化层的每一个神经元都提取了之前那个卷积层的一个2×2区域的信息。更为具体的一个例子，一种非常常见的池化操作叫做Max-pooling。在Max-Pooling中，这个神经元选择2×2区域里激活值最大的值，如下图所示：



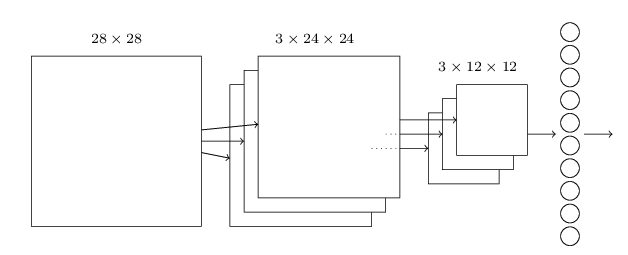
注意卷积层的输出是24×24的，而池化后是12×12的。

就像上面提到的，卷积层通常会有多个特征映射。我们会对每一个特征映射进行max-pooling操作。因此，如果一个卷积层有3个特征映射，那么卷积加max-pooling后就如下图所示：   


我们可以把max-pooling看成神经网络关心某个特征在这个区域里是否出现。它忽略了这个特征出现的具体位置。直觉上看，如果某个特征出现了，那么这个特征相对于其它特征的精确位置是不重要的【精确位置不重要，但是大致的位置是重要的，比如识别一个猫，两只眼睛和鼻子有一个大致的相对位置关系，但是在一个2×2的小区域里稍微移动一下眼睛，应该不太影响我们识别一只猫，而且它还能解决图像拍摄角度变化，扭曲等问题】。而且一个很大的好处是池化可以减少特征的个数【2×2的max-pooling让特征的大小变为原来的1/4】，因此减少了之后层的参数个数。

Max-pooling不是唯一的池化方法。另外一种常见的是L2 Pooling。这种方法不是取2×2区域的最大值，而是2×2区域的每个值平方然后求和然后取平方根。虽然细节有所不同，但思路和max-pooling是类似的：L2 Pooling也是从卷积层压缩信息的一种方法。在实践中，两种方法都被广泛使用。有时人们也使用其它的池化方法。如果你真的想尝试不同的方法来提供性能，那么你可以使用validation数据来尝试不同池化方法然后选择最合适的方法。但是这里我们不在讨论这些细节。【Max-Pooling是用的最多的，甚至也有人认为Pooling并没有什么卵用。深度学习一个问题就是很多经验的tricks由于没有太多理论依据，只是因为最早的人用了，而且看起来效果不错（但可能换一个数据集就不一定了），所以后面的人也跟着用。但是过了没多久又被认为这个trick其实没啥用】

**（4）合到一起**  
 现在我们可以把这3个idea放到一起来构建一个完整的卷积神经网络了。它和之前我们看到的结构类似，不过增加了一个有10个神经元的输出层，这个层的每个神经元对应于0-9直接的一个数字：



这个网络的输入的大小是28×28，每一个输入对于MNIST图像的一个像素。然后使用了3个特征映射，局部感知域的大小是5×5。这样得到3×24×24的输出。然后使用对每一个特征映射的输出应用2×2的max-pooling，得到3×12×12的输出。

最后一层是全连接的网络，3×12×12个神经元会连接到输出10个神经元中的每一个。这和之前介绍的全连接神经网络是一样的。

卷积结构和之前的全连接结构有很大的差别。但是整体的图景是类似的：一个神经网络有很多神经元，它们的行为有weights和biase确定。并且整体的目标也是类似的：使用训练数据来训练网络的weights和biases使得网络能够尽量好的识别图片。

和之前介绍的一样，这里我们仍然使用随机梯度下降来训练。不过反向传播算法有所不同。原因是之前bp算法的推导是基于全连接的神经网络。不过幸运的是求卷积和max-pooling的导数是非常简单的。

## 3、详解CNN的层次结构

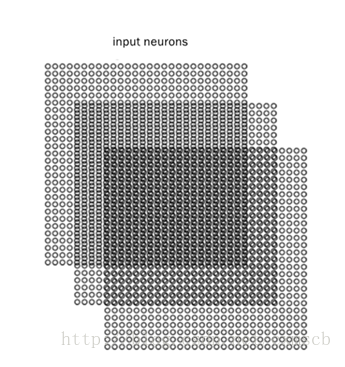
### （1）输入层

在CNN的输入层中，（图片）数据输入的格式与全连接神经网络的输入格式（一维向量）不太一样。CNN的输入层的输入格式保留了图片本身的结构。

对于黑白的 28×28 的图片，CNN的输入是一个 28×28 的的二维神经元，如下图所示：



而对于RGB格式的28×28图片，CNN的输入则是一个 3×28×28 的三维神经元（RGB中的每一个颜色通道都有一个 28×28 的矩阵），如下图所示：



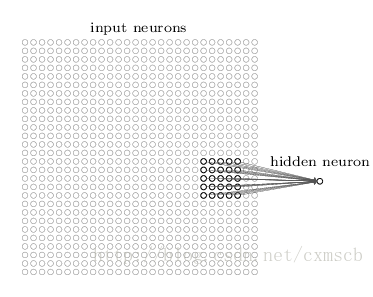
### （2）卷积层

在卷积层中有几个重要的概念：

A）local receptive fields（局部感知域）

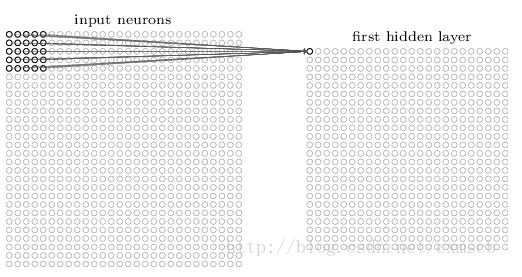
B）shared weights（共享权值）

假设输入的是一个 28×28 的的二维神经元，我们定义5×5 的一个 local receptive fields（局部感知域），即隐藏层的神经元与输入层的5×5个神经元相连，这个5\*5的区域就称之为Local Receptive Fields，如下图所示：



可类似看作：隐藏层中的神经元 具有一个固定大小的感受视野去感受上一层的部分特征。在全连接神经网络中，隐藏层中的神经元的感受视野足够大乃至可以看到上一层的所有特征。

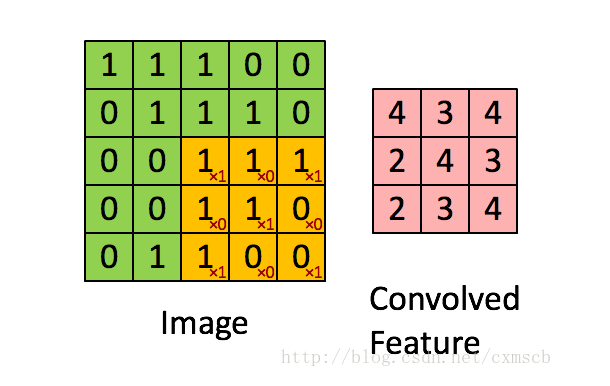
而在卷积神经网络中，隐藏层中的神经元的感受视野比较小，只能看到上一次的部分特征，上一层的其他特征可以通过平移感受视野来得到同一层的其他神经元，由同一层其他神经元来看：

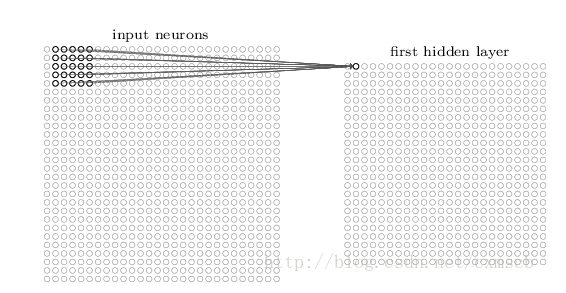


设移动的步长为1：从左到右扫描，每次移动 1 格，扫描完之后，再向下移动一格，再次从左到右扫描。

具体过程如动图所示：





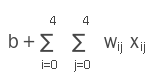
、

可看出卷积层的神经元是只与前一层的部分神经元节点相连，每一条相连的线对应一个权重 w 。

一个感受视野带有一个卷积核，我们将感受视野中的权重 w 矩阵称为卷积核；将感受视野对输入的扫描间隔称为步长（stride）；当步长比较大时（stride>1），为了扫描到边缘的一些特征，感受视野可能会“出界”，这时需要对边界扩充(pad)，边界扩充可以设为 0 或 其他值。步长和边界扩充值的大小由用户来定义。

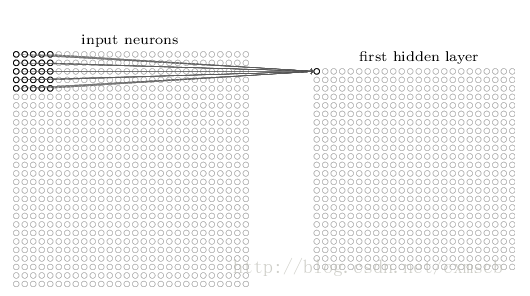
卷积核的大小由用户来定义，即定义的感受视野的大小；卷积核的权重矩阵的值，便是卷积神经网络的参数，为了有一个偏移项 ，卷积核可附带一个偏移项 b ，它们的初值可以随机来生成，可通过训练进行变化。

因此 感受视野 扫描时可以计算出下一层神经元的值为：

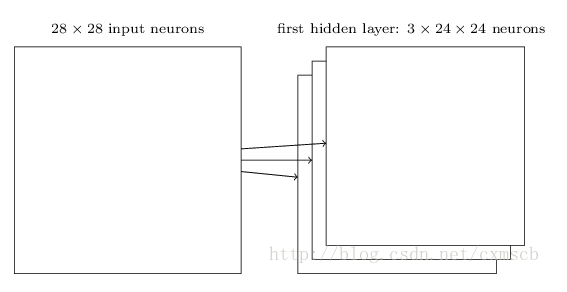


对下一层的所有神经元来说，它们从不同的位置去探测了上一层神经元的特征。

我们将通过 一个带有卷积核的感受视野 扫描生成的下一层神经元矩阵称为一个feature map (特征映射图)，如下图的右边便是一个 feature map：

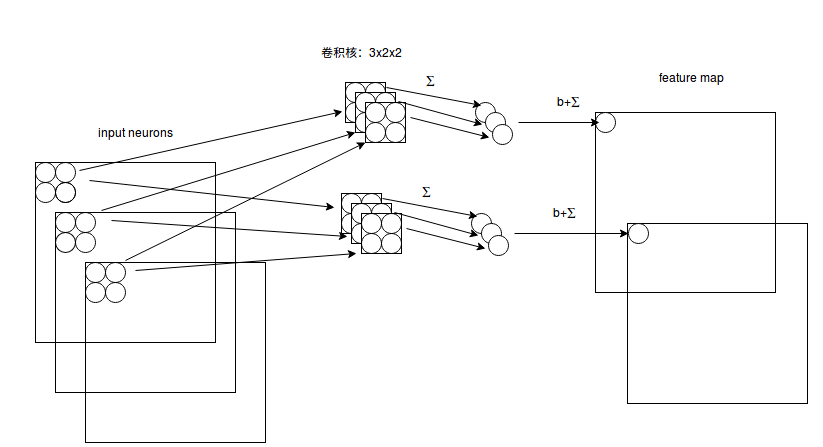


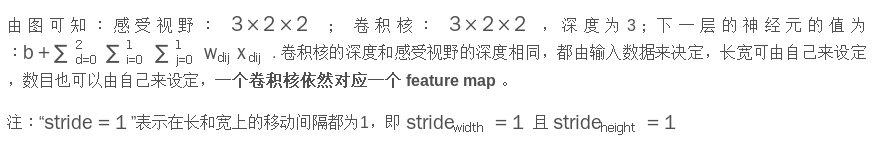
因此在同一个 feature map 上的神经元使用的卷积核是相同的，因此这些神经元 shared weights，共享卷积核中的权值和附带的偏移。一个 feature map 对应 一个卷积核，若我们使用 3 个不同的卷积核，可以输出3个feature map：（感受视野：5×5，布长stride：1）



因此在CNN的卷积层，我们需要训练的参数大大地减少到了 (5×5+1)×3=78个。

假设输入的是 28×28 的RGB图片，即输入的是一个 3×28×28 的的二维神经元，这时卷积核的大小不只用长和宽来表示，还有深度，感受视野也对应的有了深度，如下图所示：





### （3）激励层

激励层主要对卷积层的输出进行一个非线性映射，因为卷积层的计算还是一种线性计算。使用的激励函数一般为ReLu函数：

f(x)=max(x,0)

卷积层和激励层通常合并在一起称为“卷积层”。

### （4）池化层

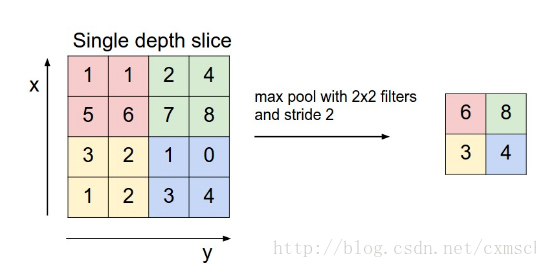
当输入经过卷积层时，若感受视野比较小，步长stride比较小，得到的feature map （特征图）还是比较大，可以通过池化层来对每一个 feature map 进行降维操作，输出的深度还是不变的，依然为 feature map 的个数。

池化层也有一个“池化视野（filter）”来对feature map矩阵进行扫描，对“池化视野”中的矩阵值进行计算，一般有两种计算方式：

A）Max pooling：取“池化视野”矩阵中的最大值

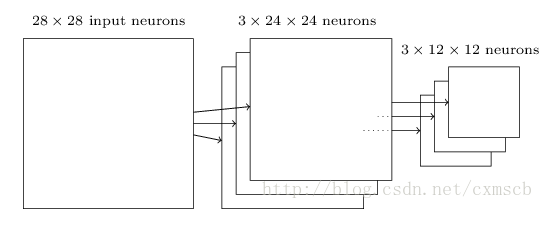
B）Average pooling：取“池化视野”矩阵中的平均值

扫描的过程中同样地会涉及的扫描步长stride，扫描方式同卷积层一样，先从左到右扫描，结束则向下移动步长大小，再从左到右。如下图示例所示：



其中“池化视野”filter： 2×2；步长stride：2。（注：“ 池化视野”为个人叫法）

最后可将 3 个 24×24 的 feature map下采样得到 3 个 24×24 的特征矩阵：

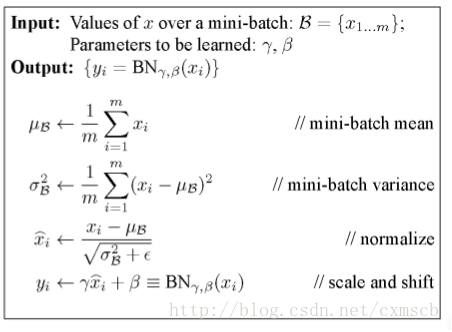


### （5）归一化层

A）Batch Normalization

Batch Normalization（批量归一化）实现了在神经网络层的中间进行预处理的操作，即在上一层的输入归一化处理后再进入网络的下一层，这样可有效地防止“梯度弥散”，加速网络训练。、

Batch Normalization具体的算法如下图所示：



每次训练时，取 batch\_size 大小的样本进行训练，在BN层中，将一个神经元看作一个特征，batch\_size 个样本在某个特征维度会有 batch\_size 个值，然后在每个神经元 xi 维度上的进行这些样本的均值和方差，通过公式得到 xi∧，再通过参数 γ 和 β 进行线性映射得到每个神经元对应的输出 yi 。在BN层中，可以看出每一个神经元维度上，都会有一个参数 γ 和 β ，它们同权重w一样可以通过训练进行优化。

在卷积神经网络中进行批量归一化时，一般对未进行ReLu激活的 feature map进行批量归一化，输出后再作为激励层的输入，可达到调整激励函数偏导的作用。

一种做法是将 feature map 中的神经元作为特征维度，参数 γ 和 β 的数量和则等于 2×fmapwidth×fmaplength×fmapnum，这样做的话参数的数量会变得很多；

另一种做法是把 一个 feature map 看做一个特征维度，一个 feature map 上的神经元共享这个 feature map的 参数 γ 和 β ，参数 γ 和 β 的数量和则等于 2×fmapnum，计算均值和方差则在batch\_size个训练样本在每一个feature map维度上的均值和方差。

注：fmapnum指的是一个样本的feature map数量，feature map 跟神经元一样也有一定的排列顺序。

**Batch Normalization 算法的训练过程和测试过程的区别：**

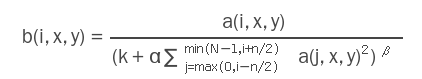
在训练过程中，我们每次都会将 batch\_size 数目大小的训练样本 放入到CNN网络中进行训练，在BN层中自然可以得到计算输出所需要的均值和方差 ;

而在测试过程中，我们往往只会向CNN网络中输入一个测试样本，这时在BN层计算的均值和方差会均为0，因为只有一个样本输入，因此BN层的输入也会出现很大的问题，从而导致CNN网络输出的错误。所以在测试过程中，我们需要借助训练集中所有样本在BN层归一化时每个维度上的均值和方差，当然为了计算方便，我们可以在 batch\_num 次训练过程中，将每一次在BN层归一化时每个维度上的均值和方差进行相加，最后再进行求一次均值即可。

B）Local Response Normalization

近邻归一化(Local Response Normalization)的归一化方法主要发生在不同的相邻的卷积核（经过ReLu之后）的输出之间，即输入是发生在不同的经过ReLu之后的 feature map 中。

LRN的公式如下：、



其中：

a(i,x,y) 表示第i个卷积核的输出（经过ReLu层）的feature map上的 (x，y) 位置上的值。

b(i,x,y) 表示 a(i,x,y) 经LRN后的输出。

N 表示卷积核的数量，即输入的 feature map的个数。

n 表示近邻的卷积核（或feature map）个数，由自己来决定。

k,α,β是超参数，由用户自己调整或决定。

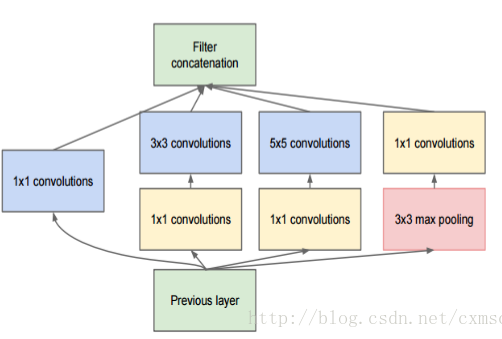
与BN的区别：BN依据mini batch的数据,近邻归一仅需要自己来决定，BN训练中有学习参数;BN归一化主要发生在不同的样本之间，LRN归一化主要发生在不同的卷积核的输出之间。

### （6）切分层：

在一些应用中,需要对图片进行切割，独立地对某一部分区域进行单独学习。这样可以对特定部分进行通过调整 感受视野 进行力度更大的学习。

### （7）融合层：

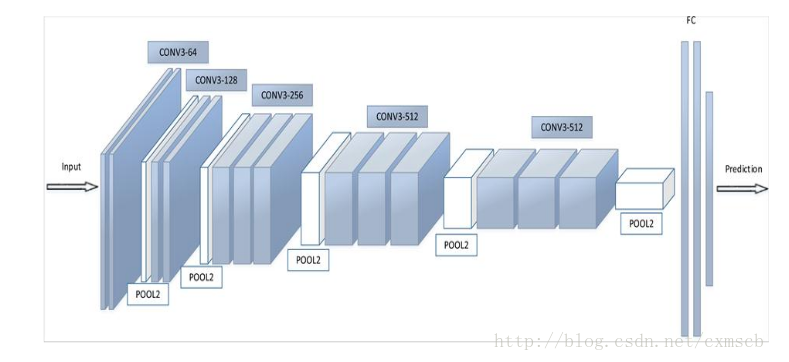
融合层可以对切分层进行融合，也可以对不同大小的卷积核学习到的特征进行融合。例如在GoogleLeNet 中，使用多种分辨率的卷积核对目标特征进行学习，通过 padding 使得每一个 feature map 的长宽都一致，之后再将多个 feature map 在深度上拼接在一起：



融合的方法有几种，一种是特征矩阵之间的拼接级联，另一种是在特征矩阵进行运算 (+,−,x,max,conv)。

### （8）全连接层和输出层

全连接层主要对特征进行重新拟合，减少特征信息的丢失；输出层主要准备做好最后目标结果的输出。例如VGG的结构图，如下图所示：

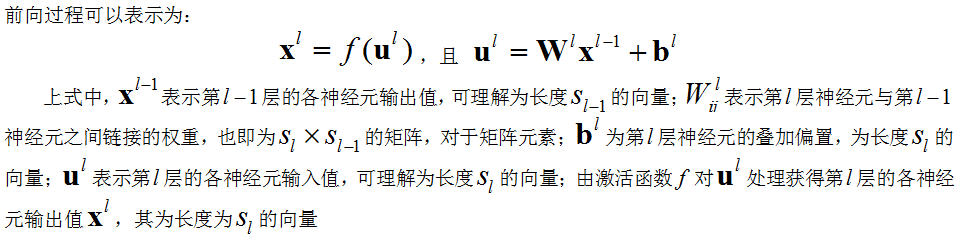


## 4、CNN训练过程

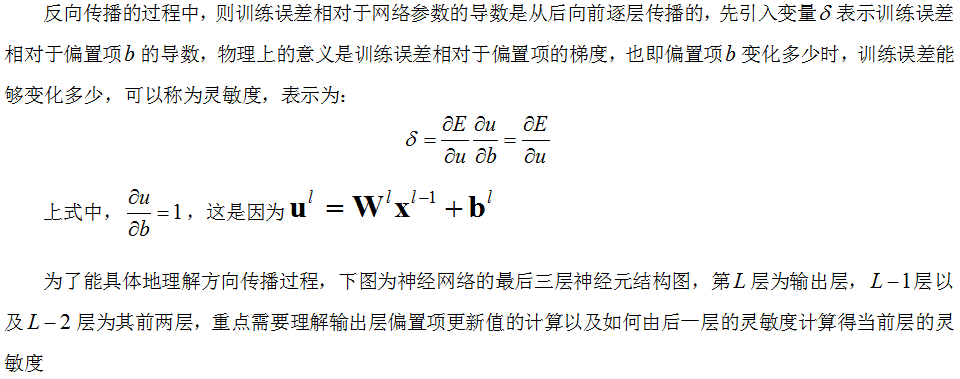
### 4.1神经网络BP算法

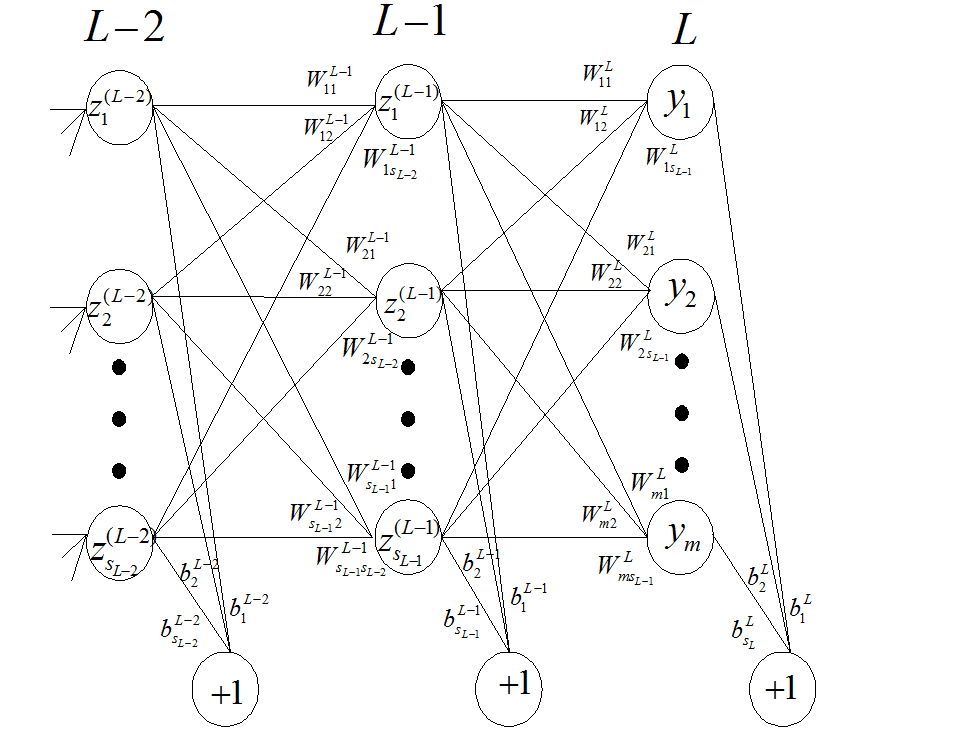


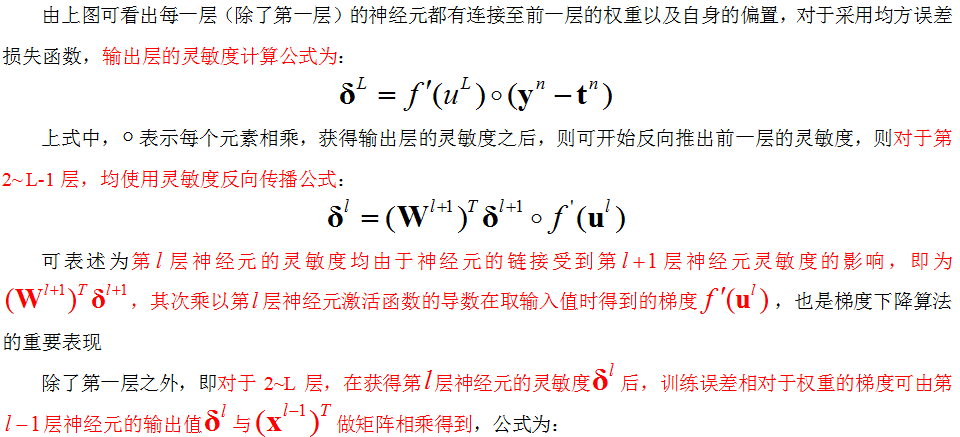
（1）前向传播过程

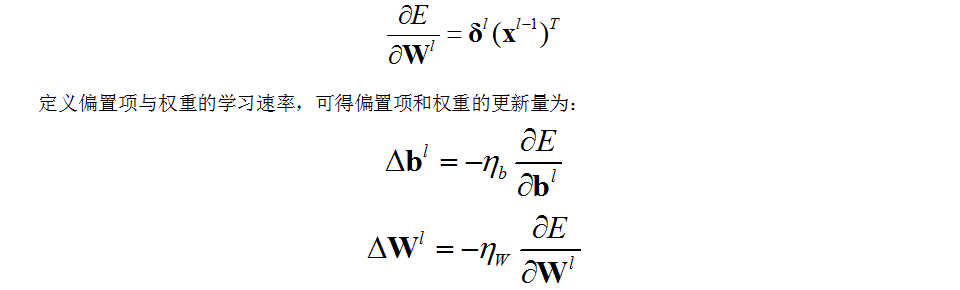


（2）反向传播过程

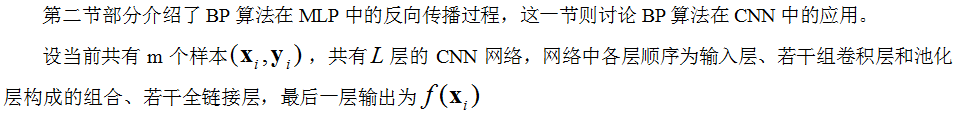




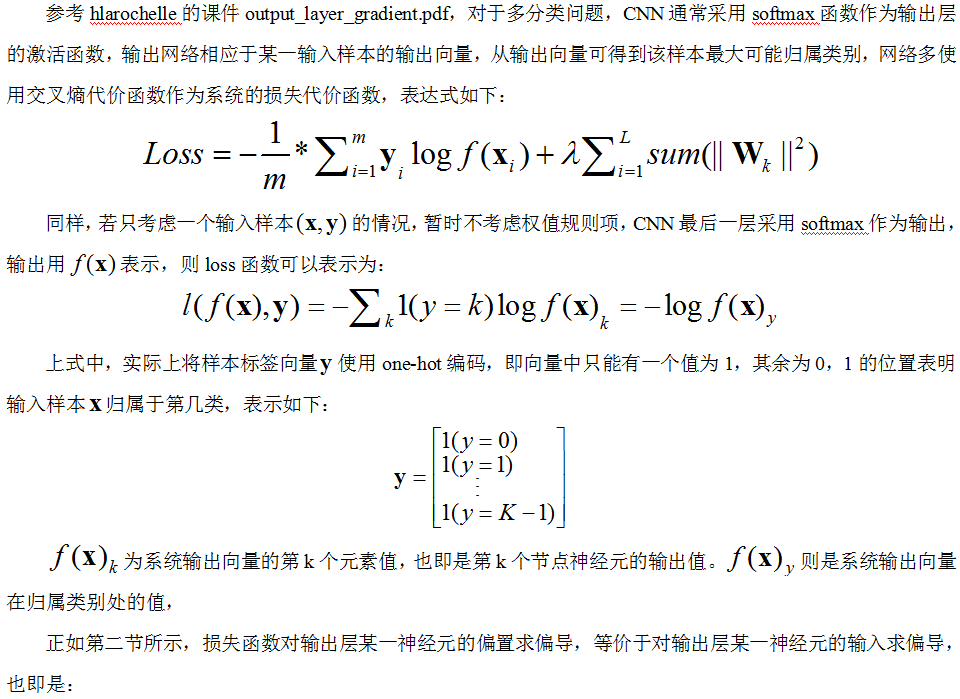


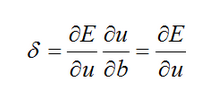


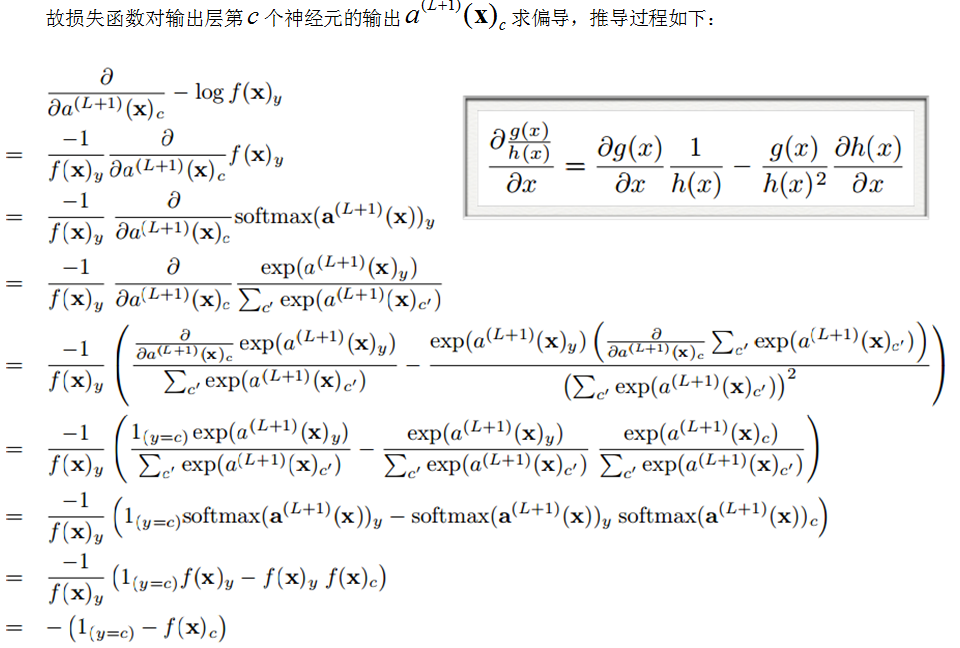
### 4.2 BP算法在CNN中的应用

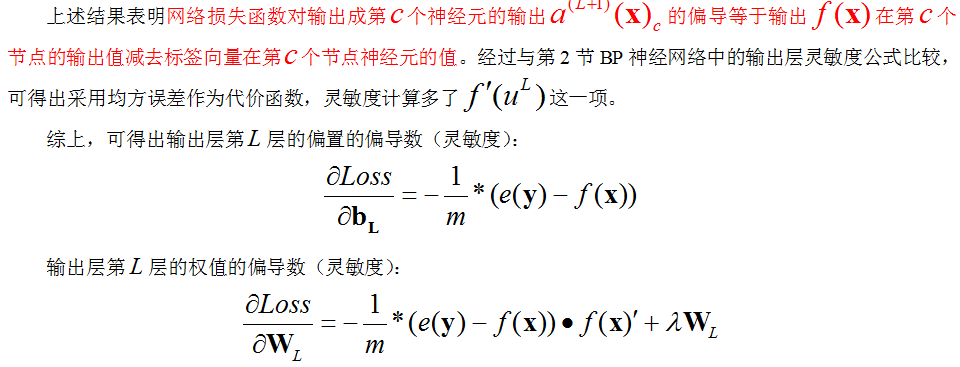


（1）采用softmax交叉熵代价函数时输出层灵敏度计算





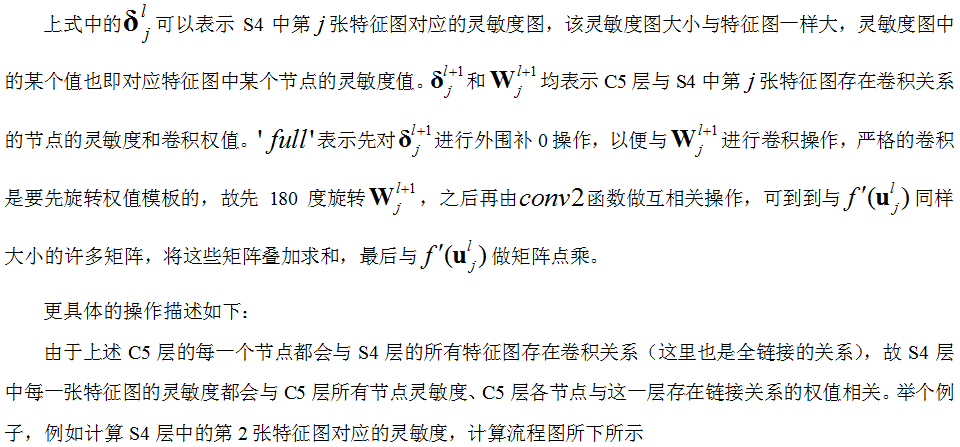


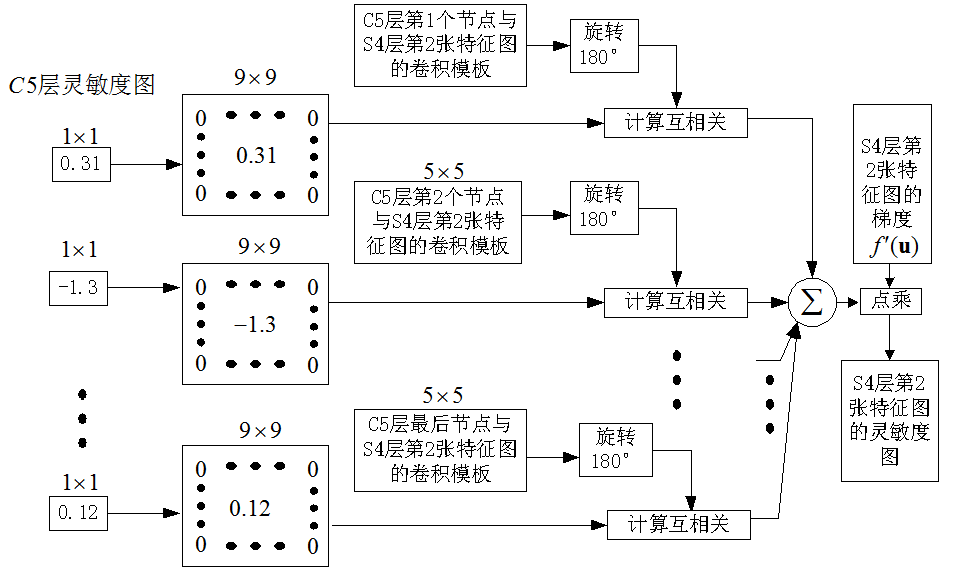


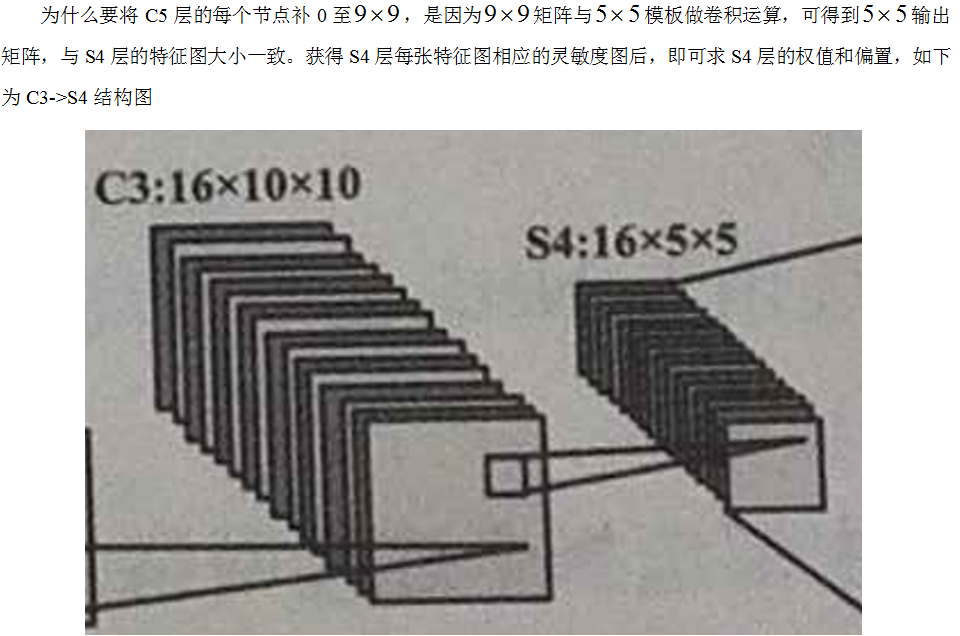
（2）全连接层的前一层为池化层时，求池化层的灵敏度及权重和偏置的梯度

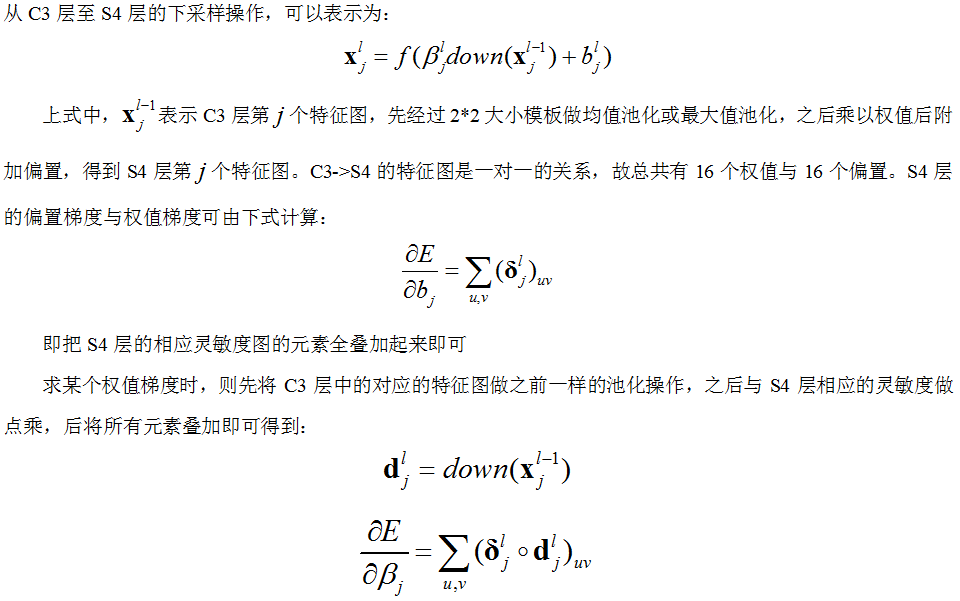
2018-03-02 15-49-48 的屏幕截图



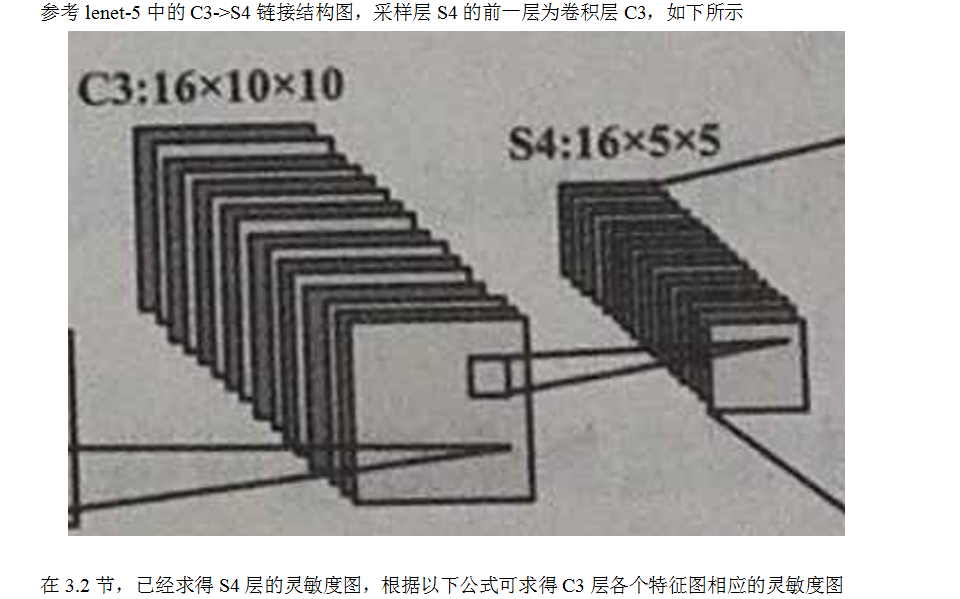


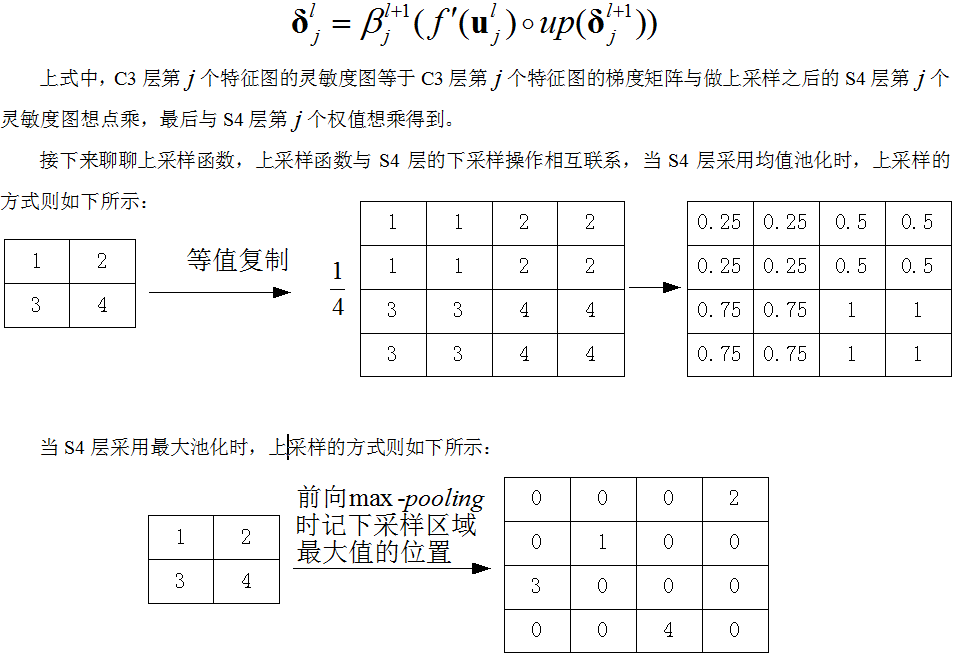


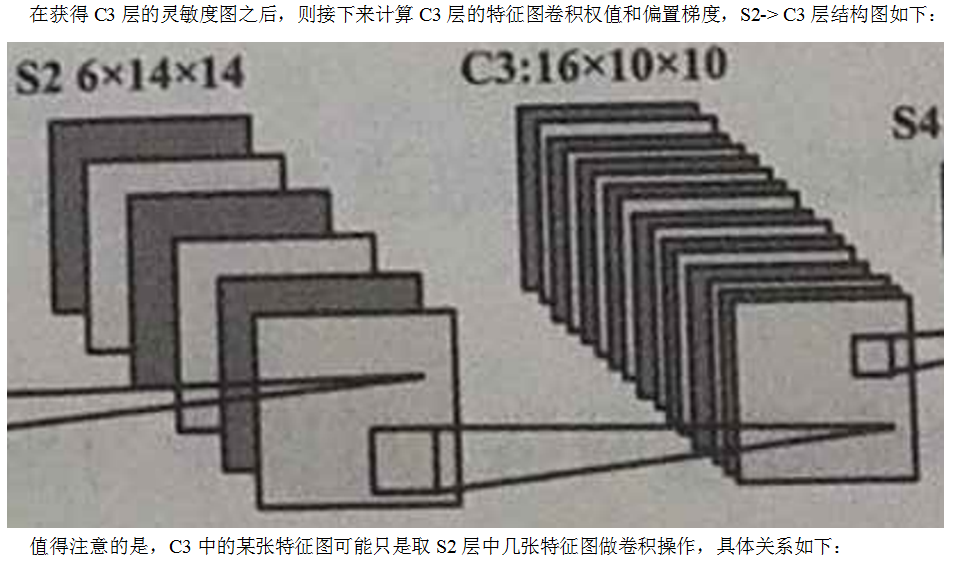


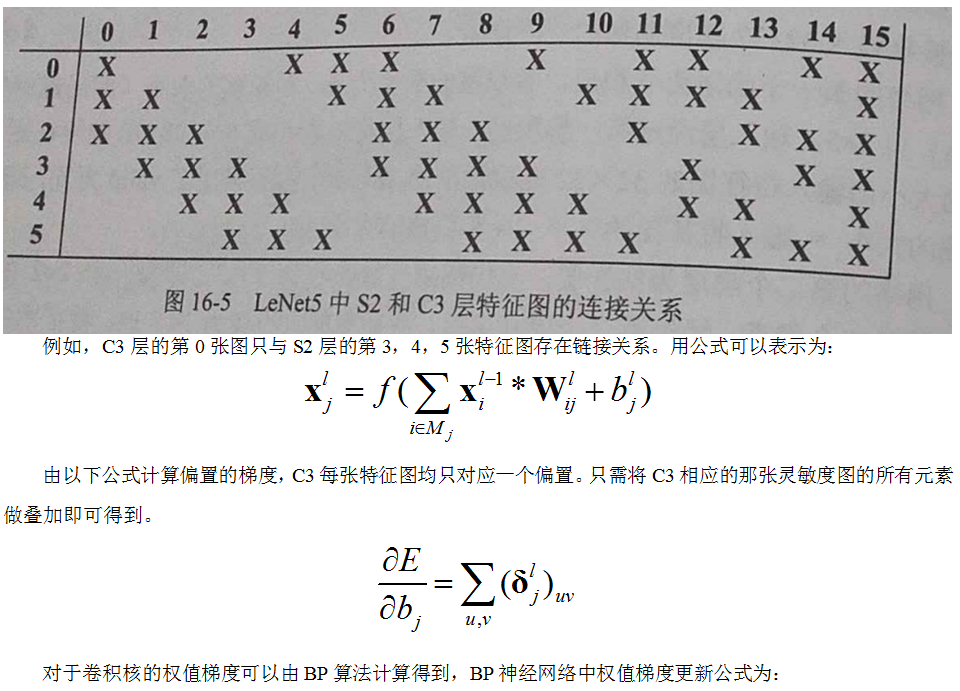


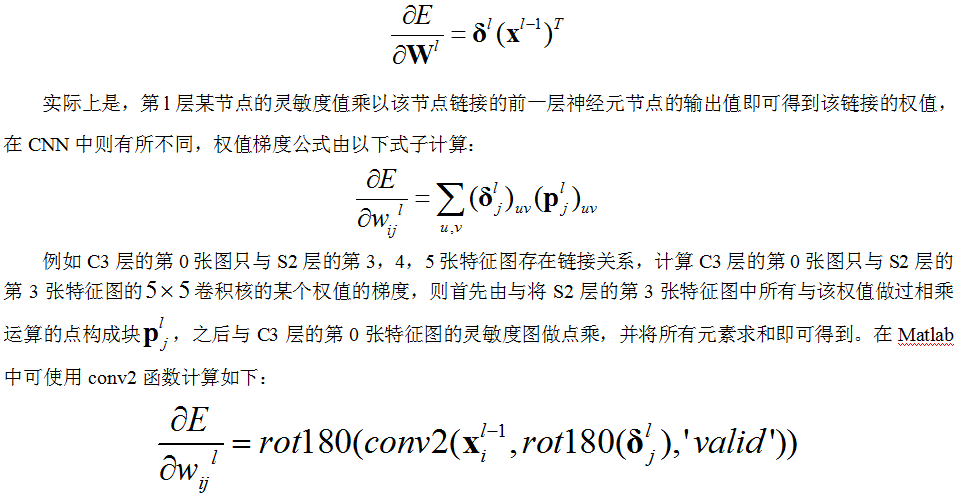
（3）池化层的前一层为卷积层时，求卷积层的灵敏度及权重与偏置的梯度



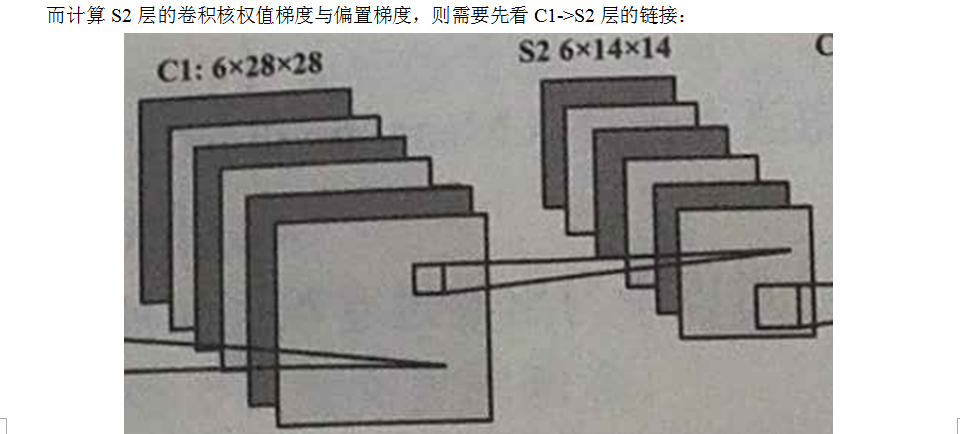
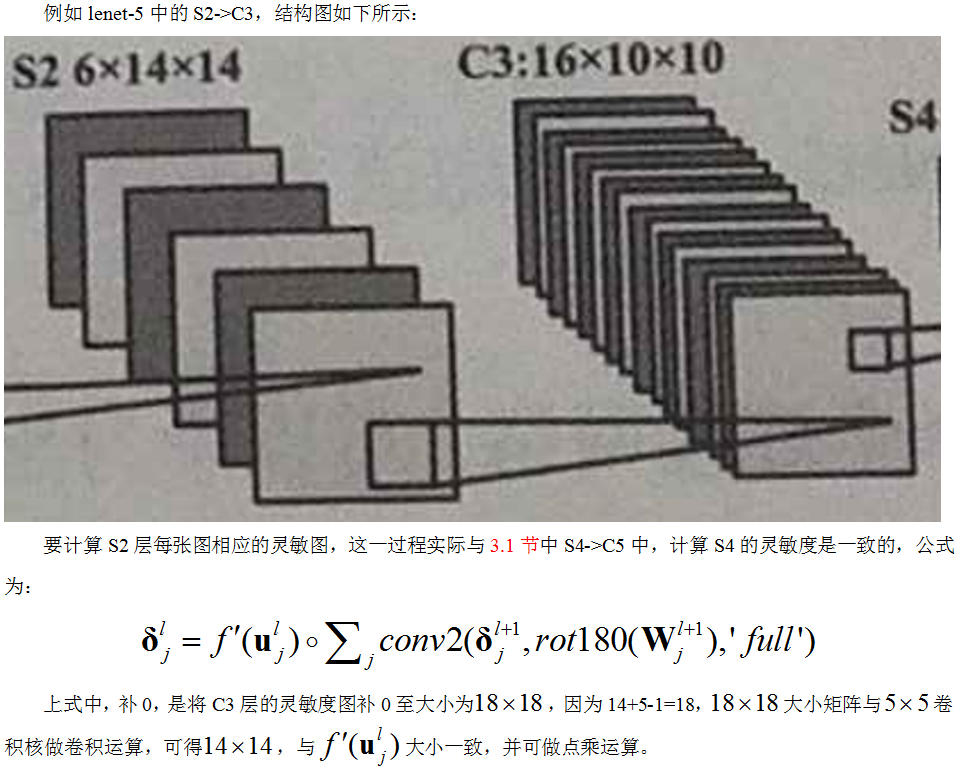


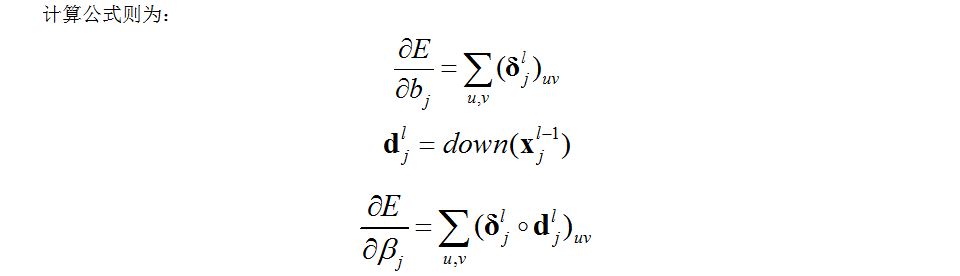






（4）卷积层的前一层为池化层时，求卷积层灵敏度及卷积核权值梯度与偏置梯度





## 5、典型卷积网络

**LeNet:《Gradient-Based Learning Applied to Document Recognition》** CNN的开山之作，也是手写体识别经典论文。

**AlexNet:《 ImageNet Classification with Deep Convolutional Neural Networks 》** ILSVRC-2012 大赛冠军，促进CNN的扛鼎之作，AlexNet是CNN发展史上的一个历史性转折，不能不读。

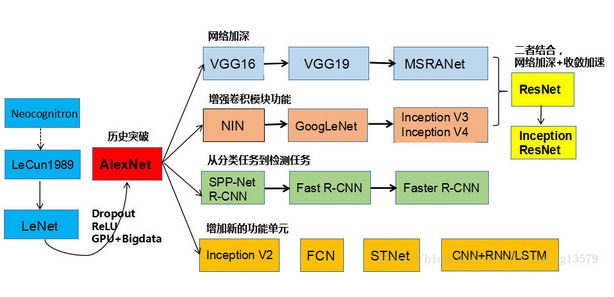
**Inception V1和V3:《Going Deeper with Convolutions》,《Rethinking the Inception Architecture for Computer Vision》**,2014年ImageNet大赛冠军,Inception结构的设计很巧妙

**VGGNet:《Very Deep Convolutional Networks for Large-Scale Image Recognition》**,虽然不是那年ImageNet大赛的冠军(那年的冠军是GoogLeNet),但是VGGNet对后面的ResNet，Inception产生了重要的影响

**DeepID2+:《Deeply learned face representations are sparse, selective, and robust》**为什么要推荐这篇论文呢？人脸识别领域，DeepID大名如雷贯耳，与DeepID,DeepID2不同的是，这篇论文并不是单纯讲人脸识别，论文深入分析了CNN的内部结构，试图从理论上解释CNN强大的特征提取能和分类识别能力，这是学者第一次试图去探索CNN的本质属性，看完这篇论文，相信对CNN会有更深入的了解。

**ResNet:《Deep Residual Learning for Image Recognition》**，直接将top5错误率降到了3.57%（GoogLeNet 是6.66%），超越了人眼，文中最大的亮点就是残差块结构的设计。

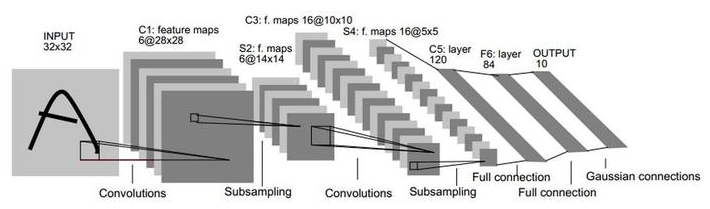
这里附上一张CNN结构演化历史的图，希望帮助大家更好的了解CNN的发展。



### （1）LeNet-5

LeNet-5共有7层，不包含输入，每层都包含可训练参数（连接权重）。输入图像为32\*32大小。这要比Mnist数据库（一个公认的手写数据库）中最大的字母还大。这样做的原因是希望潜在的明显特征如笔画断点或角点能够出现在最高层特征监测子感受视野（局部感知域）的中心。

我们先要明确一点：每个层有多个Feature Map，每个Feature Map通过一种卷积滤波器提取输入的一种特征，然后每个Feature Map有多个神经元。



A） C1层

C1层是一个卷积层（为什么是卷积？卷积运算一个重要的特点就是，通过卷积运算，可以使原信号特征增强，并且降低噪音），由6个特征图Feature Map构成。特征图中每个神经元与输入中5\*5的邻域相连。特征图的大小为28\*28。C1有156个可训练参数（每个滤波器5\*5=25个unit参数和一个bias参数，一共6个滤波器，共(5\*5+1)\*6=156个参数），共156\*(28\*28)=122,304个连接。

输入图片：32\*32

卷积核大小：5\*5

卷积核种类：6

输出featuremap大小：28\*28 （32-5+1）

神经元数量：28\*28\*6

可训练参数：（5\*5+1）\*6（每个滤波器5\*5=25个unit参数和一个bias参数，一共6个滤波器）

连接数：（5\*5+1）\*6\*28\*28

B）S2层

S2层是一个下采样层（为什么是下采样？利用图像局部相关性的原理，对图像进行子抽样，可以减少数据处理量同时保留有用信息），有6个14\*14的特征图。特征图中的每个单元与C1中相对应特征图的2\*2邻域相连接。S2层每个单元的4个输入相加，乘以一个可训练参数，再加上一个可训练偏置。结果通过sigmoid函数计算。可训练系数和偏置控制着sigmoid函数的非线性程度。如果系数比较小，那么运算近似于线性运算，亚采样相当于模糊图像。如果系数比较大，根据偏置的大小亚采样可以被看成是有噪声的“或”运算或者有噪声的“与”运算。每个单元的2\*2感受野并不重叠，因此S2中每个特征图的大小是C1中特征图大小的1/4（行和列各1/2）。S2层有12个可训练参数和5880个连接。

输入：28\*28

采样区域：2\*2

采样方式：4个输入相加，乘以一个可训练参数，再加上一个可训练偏置。结果通过sigmoid

采样种类：6

输出featureMap大小：14\*14（28/2）

神经元数量：14\*14\*6

可训练参数：2\*6（和的权+偏置）

连接数：（2\*2+1）\*6\*14\*14

S2中每个特征图的大小是C1中特征图大小的1/4

C）C3层

C3层也是一个卷积层，它同样通过5x5的卷积核去卷积层S2，然后得到的特征map就只有10x10个神经元，但是它有16种不同的卷积核，所以就存在16个特征map了。这里需要注意的一点是：C3中的每个特征map是连接到S2中的所有6个或者几个特征map的，表示本层的特征map是上一层提取到的特征map的不同组合（这个做法也并不是唯一的）。

输入：S2中所有6个或者几个特征map组合

卷积核大小：5\*5

卷积核种类：16

输出featureMap大小：10\*10

C3中的每个特征map是连接到S2中的所有6个或者几个特征map的，表示本层的特征map是上一层提取到的特征map的不同组合

存在的一个方式是：C3的前6个特征图以S2中3个相邻的特征图子集为输入。接下来6个特征图以S2中4个相邻特征图子集为输入。3个以不相邻的4个特征图子集为输入。最后一个将S2中所有特征图为输入。

则：可训练参数：6\*（3\*25+1）+6\*（4\*25+1）+3\*（4\*25+1）+（25\*6+1）=1516

连接数：10\*10\*1516=151600

D）S4层

S4层是一个下采样层，由16个5\*5大小的特征图构成。特征图中的每个单元与C3中相应特征图的2\*2邻域相连接，跟C1和S2之间的连接一样。S4层有32个可训练参数（每个特征图1个因子和一个偏置 2\*16）和2000个连接。

输入：10\*10

采样区域：2\*2

采样方式：4个输入相加，乘以一个可训练参数，再加上一个可训练偏置。结果通过sigmoid

采样种类：16

输出featureMap大小：5\*5（10/2）

神经元数量：5\*5\*16=400

可训练参数：2\*16=32（和的权+偏置）

连接数：16\*（2\*2+1）\*5\*5=2000

S4中每个特征图的大小是C3中特征图大小的1/4

E）C5层

C5层是一个卷积层，有120个特征图。每个单元与S4层的全部16个单元的5\*5邻域相连。由于S4层特征图的大小也为5\*5（同滤波器一样），故C5特征图的大小为1\*1：这构成了S4和C5之间的全连接。之所以仍将C5标示为卷积层而非全连接层，是因为如果LeNet-5的输入变大，而其他的保持不变，那么此时特征图的维数就会比1\*1大。C5层有48120个可训练连接。

输入：S4层的全部16个单元特征map（与s4全相连）

卷积核大小：5\*5

卷积核种类：120

输出featureMap大小：1\*1（5-5+1）

可训练参数/连接：120\*（16\*5\*5+1）=48120

F）F6层

F6层有84个单元？（之所以选这个数字的原因来自于输出层的设计），与C5层全相连。有10164个可训练参数。如同经典神经网络，F6层计算输入向量和权重向量之间的点积，再加上一个偏置。然后将其传递给sigmoid函数产生单元i的一个状态。

输入：c5 120维向量

计算方式：计算输入向量和权重向量之间的点积，再加上一个偏置，结果通过sigmoid函数

可训练参数:84\*(120+1)=10164

G）输出层

最后，输出层由欧式径向基函数（Euclidean Radial Basis Function）单元组成，每类一个单元，每个有84个输入。换句话说，每个输出RBF单元计算输入向量和参数向量之间的欧式距离。输入离参数向量越远，RBF输出的越大。一个RBF输出可以被理解为衡量输入模式和与RBF相关联类的一个模型的匹配程度的惩罚项。用概率术语来说，RBF输出可以被理解为F6层配置空间的高斯分布的负log-likelihood（对数似然）。给定一个输入模式，损失函数应能使得F6的配置与RBF参数向量（即模式的期望分类）足够接近。这些单元的参数是人工选取并保持固定的（至少初始时候如此）。这些参数向量的成分被设为-1或1。虽然这些参数可以以-1和1等概率的方式任选，或者构成一个纠错码，但是被设计成一个相应字符类的7\*12大小（即84）的格式化图片。这种表示对识别单独的数字不是很有用，但是对识别可打印ASCII集中的字符串很有用。

使用这种分布编码而非更常用的“1 of N”编码用于产生输出的另一个原因是，当类别比较大的时候，非分布编码的效果比较差。原因是大多数时间非分布编码的输出必须为0。这使得用sigmoid单元很难实现。另一个原因是分类器不仅用于识别字母，也用于拒绝非字母。使用分布编码的RBF更适合该目标。因为与sigmoid不同，他们在输入空间的较好限制的区域内兴奋，而非典型模式更容易落到外边。

RBF参数向量起着F6层目标向量的角色。需要指出这些向量的成分是+1或-1，这正好在F6 sigmoid的范围内，因此可以防止sigmoid函数饱和。实际上，+1和-1是sigmoid函数的最大弯曲的点处。这使得F6单元运行在最大非线性范围内。必须避免sigmoid函数的饱和，因为这将会导致损失函数较慢的收敛和病态问题。

### （2）VGG

A）导引

VGGNet是2014年ILSVRC竞赛的第二名，没错你没听错它是第二名，第一名是GoogLeNet（真不是我打错google，是谷歌为了纪念LeNet，所以用的大写L）.为什么先讲VGG，因为它这个模型在多个迁移学习任务中的表现要优于googLeNet。而且，从图像中提取CNN特征，VGG模型是首选算法。它的缺点在于，参数量有140M之多，需要更大的存储空间。但是这个模型很有研究价值。

为什么叫VGG？

是牛津大学 Visual Geometry Group（视觉几何组）的同志写的论文，所以叫VGG.

B）模型解读

这张图的意思是他们一共建了A, B, C, D, E, F 6个不同的网络进行效果的比对。

注：在你看这里的时候我已经假设你看懂了AlexNet，已经对神经网络的结构有了个大致的印象。

结构A：和AlexNet类似，卷积层分为了5个stage，全连接层还是3层。只不过卷积层用的都是3x3大小的filter，具体的细节我会在下文接着阐述。

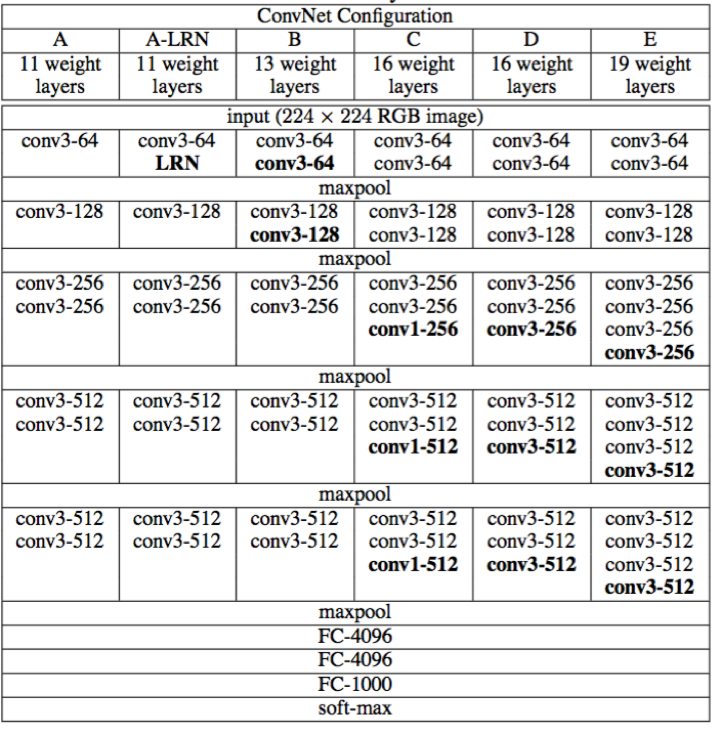
结构A-LRN：保留AlexNet里面LRN操作，其他与结构A无区别。

结构B：在A的stage2和stage3分别增加一个3x3的卷积层，共有10个卷积层。

结构C：在B的基础上，stage3，stage4，stage5分别增加一个1x1的卷积层，有13个卷积层，总计16层。

结构D：在C的基础上，stage3，stage4，stage5分别增加一个3x3的卷积层，有13个卷积层，总计16层。

结构E：在D的基础上，stage3，stage4，stage5分别再增加一个3x3的卷积层，有16个卷积层，总计19层。



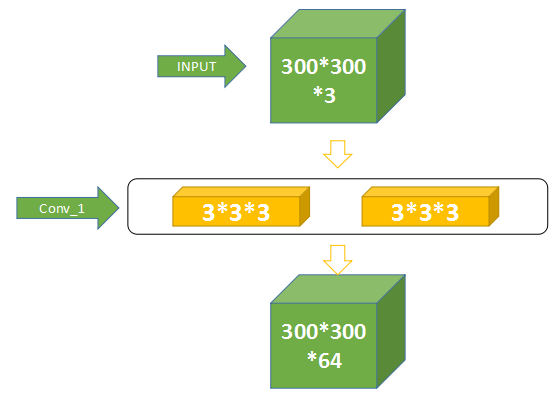
看懂一些式子表达：

Conv3-512 → 第三层卷积后维度变成512；

Conv3\_2 s=2 → 第三层卷积层里面的第二子层，滑动步长等于2（每次移动两个格子）

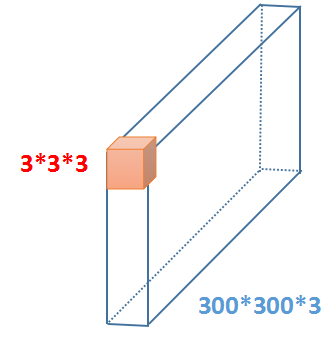
好了，我们有了以上的知识可以考试剖析VGG16卷积神经网络了。

**【1、从INPUT到Conv1：】**



首先两个黄色的是卷积层，是VGG16网络结构十六层当中的第一层（Conv1\_1）和第二层（Conv1\_2），他们合称为Conv1。

我们主要讲述第一个，也就是第一层（Conv1\_1），它怎么把一个300\*300\*3的矩阵变成一个300\*300\*64的矩阵？



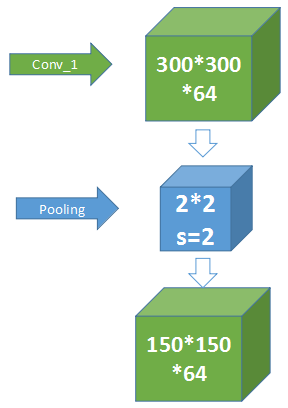
我们假设蓝色框是一个RGB图像，橙色是一个3\*3\*3的卷积核，我们对一个三维的27个数求和，然后扫过去，按照第一部分算的得出来的是一维的298\*298的矩阵（因为卷积核也是三维所以结果是一维）；

然后回想一下什么是Padding、前面也讲过它的概念了；所以不了一圈的圆，回到了300\*300\*1；

然后，VGG16这一层安置有64个卷积核，那么，原来的300\*300\*1变成300\*300\*64

于是我们的到了想要的东西；最后的绿色框；

**【2、从Conv1到Conv2之间的过度：】**



这一步用的Pooling是：2\*2\*64 s=2；

也就是说，步长是二，滑动的矩阵本身没有重叠；刚好减半，第三维度64不变；

**【3、顺利来到Conv2并且结构完全一样进入Conv3：】**

我们知道原来INPUT是300\*300\*3过了第一层出来时150\*150\*64

那么第二层仍然有池化,有128个卷积核，联想推理：

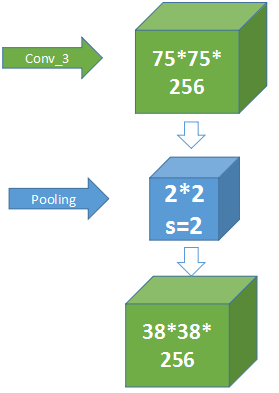
出来的应该是75\*75\*128；这一步没有问题，我们继续往下分析：

**【4、进入Conv3的推演：】**



可以知道第三层有256个卷积核，包含三层小的卷基层：

**【5、从Conv3到Conv4之间的过度：】**



池化没有问题，但是这里75不是一个偶数怎么弄，还记得我们第一部分前面的括号吗？

就是这样，我们在75这里相加了一个一，使之成为76，变成一个偶数，还有一种方法是通过步长的设置这里先不展开来讲了；